Introduction to Random Numbers, Sampling, and MCMC Methods

Bill Watson

S&P Global

August 13, 2019

What is Sampling and why is it useful?

- Sampling is the practice of generating observations from a population
- Monte Carlo methods are algorithms that rely on repeated random sampling to obtain approximations where it is difficult or impossible to use deterministic approaches
  - Optimization
  - Numerical Integration
  - Sampling distributions

Application: Approximating π

Algorithm 1 Approximating π

Input: Batch Size N

1: Sample $u_1 \sim \text{Uniform}(0, 1)$ N times

2: Sample $u_2 \sim \text{Uniform}(0, 1)$ N times

3: $\tilde{\pi} = \frac{4}{N} \cdot \left| \left\{ (u_1, u_2) \;\middle|\; \sqrt{u_1^2 + u_2^2} < 1 \right\} \right|$

Output: $\tilde{\pi}$

Application: Approximating π

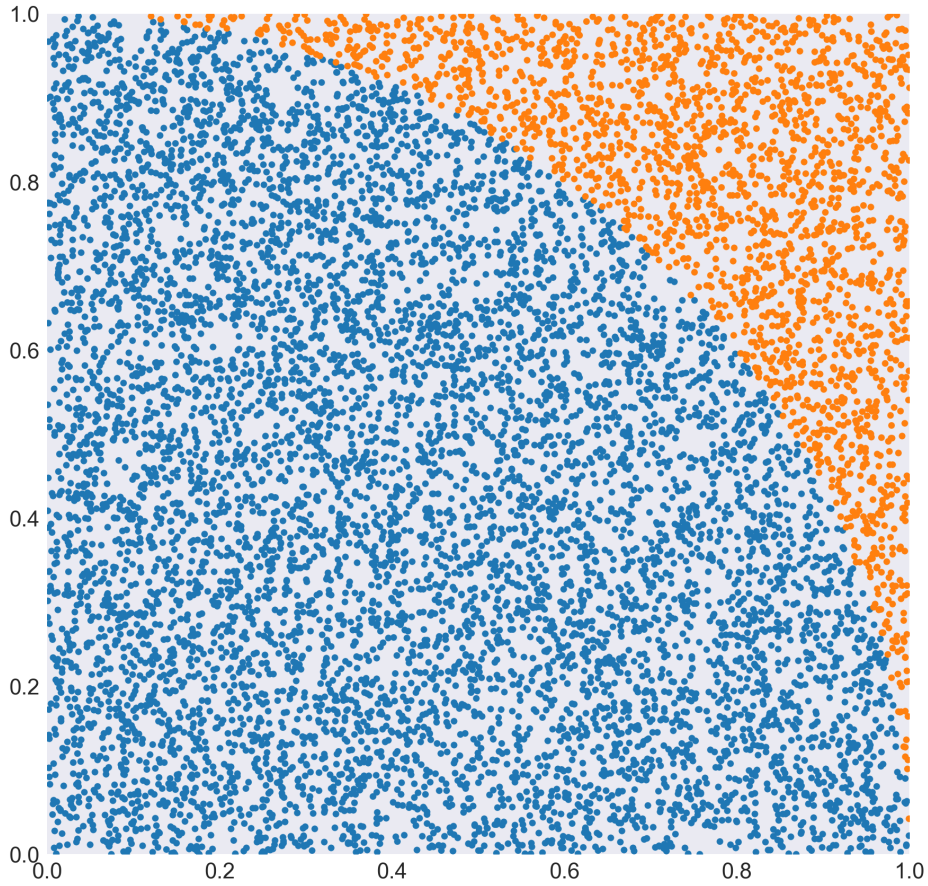
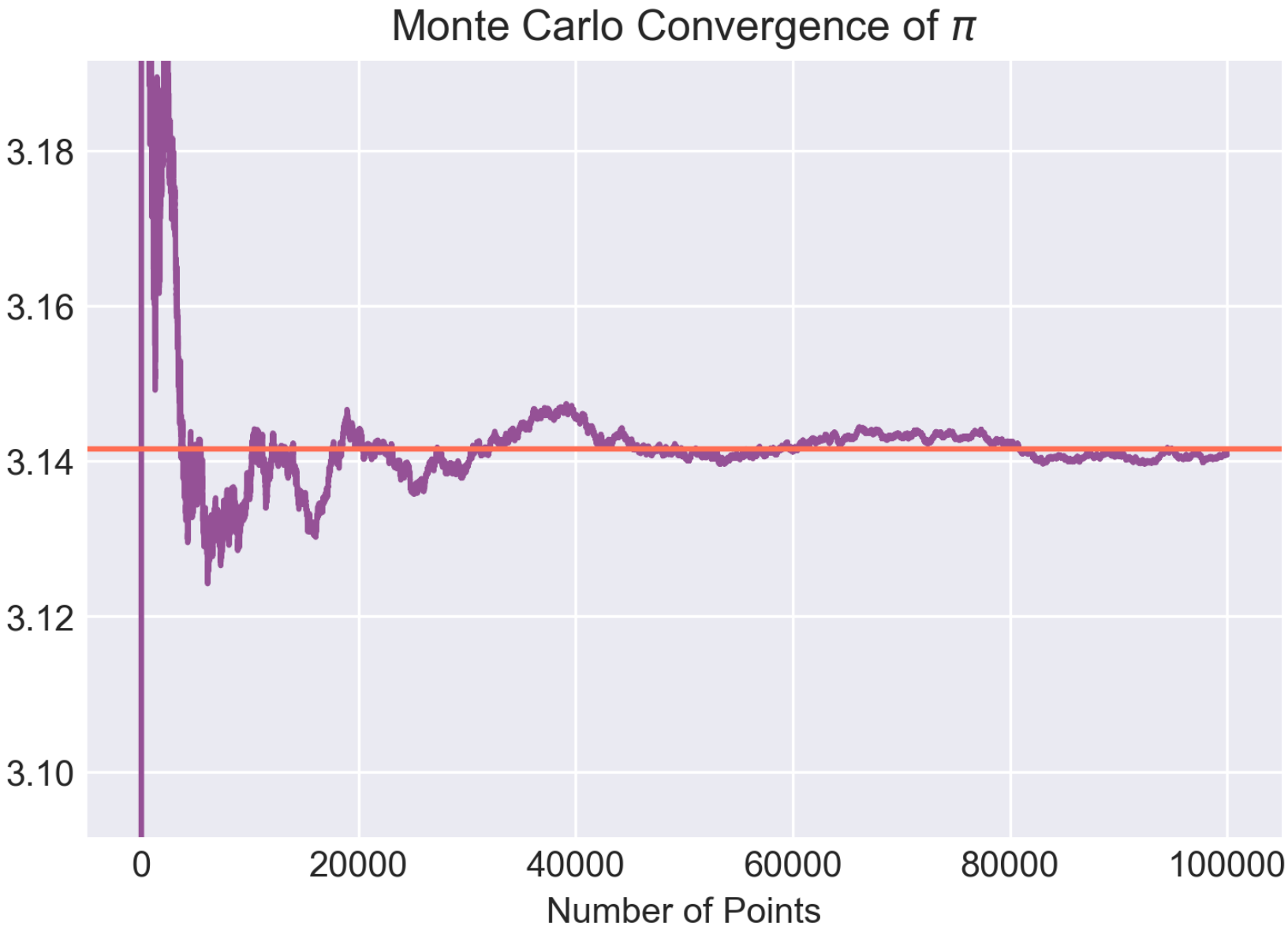


Figure: The ratio of points inside the unit circle approximates $\frac{\pi}{4}$

Application: Approximating π



What can we do with our samples?

- Integration: $\int p(x)\, f(x)\, dx \approx \frac{1}{n} \sum_{x_i \sim P} f(x_i)$
- Expectation: $\mu \approx \frac{1}{n} \sum_{x_i} x_i$
- Variance: $\sigma^2 \approx \frac{1}{n} \sum_{x_i} (x_i - \mu)^2$
- Median: $\text{median} \approx \text{median}(x_1, x_2, \ldots x_n)$
- Entropy: $\mathbb{H}(P) \approx -\frac{1}{n} \sum_{x_i \sim P} \log p(x_i)$
- CDF: $p(c) \approx \frac{1}{n} |\{x_i \,|\, x_i \leq c\}|$

Pseudo-Random Number Generators

Pseudo-Random Number Generators

- If we need random numbers, then how do we generate them?
- One solution: Pseudo-Random Number Generators
- Pseudo since they cannot simulate "true" randomness
- But can be replicated via "seeds"

Pseudo-Random Number Generators: LCG

Algorithm 2 Linear Congruential Generator

Input: Modulus m, Multiplier a, Increment c, Seed X_0

1: $X_{i+1} = (a \cdot X_i + c) \mod m$

Output: X_{i+1}

Pseudo-Random Number Generators

- LCGs are of low quality
- Mersenne Twister (1998) is the first PRNG to avoid major problems and run quickly
- Can also use Hardware (True) RNG, External Entropy PRNGs

Inverse Transform Sampling

Generating a Normal from the CDF

- We can use the cumulative distribution function to sample any distribution
- For instance, a normal's CDF is:

$$F(x) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right]$$

- With an Inverse CDF as:

$$F^{-1}(p) = \mu + \sigma\sqrt{2} \cdot \text{erf}^{-1}(2p-1)$$

- $\text{erf}(x)$ is the error function, defined as:

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

Inverse Transform Sampling

- It's easy to generalize this method to any distribution with a closed-form inverse CDF

Algorithm 3 Inverse Transform Sampling

Input: Inverse CDF F^{-1}

1: Sample $u \sim \text{Uniform}(0, 1)$

2: $X = F^{-1}(u)$

Output: X

Inverse Transform Sampling: Intuition

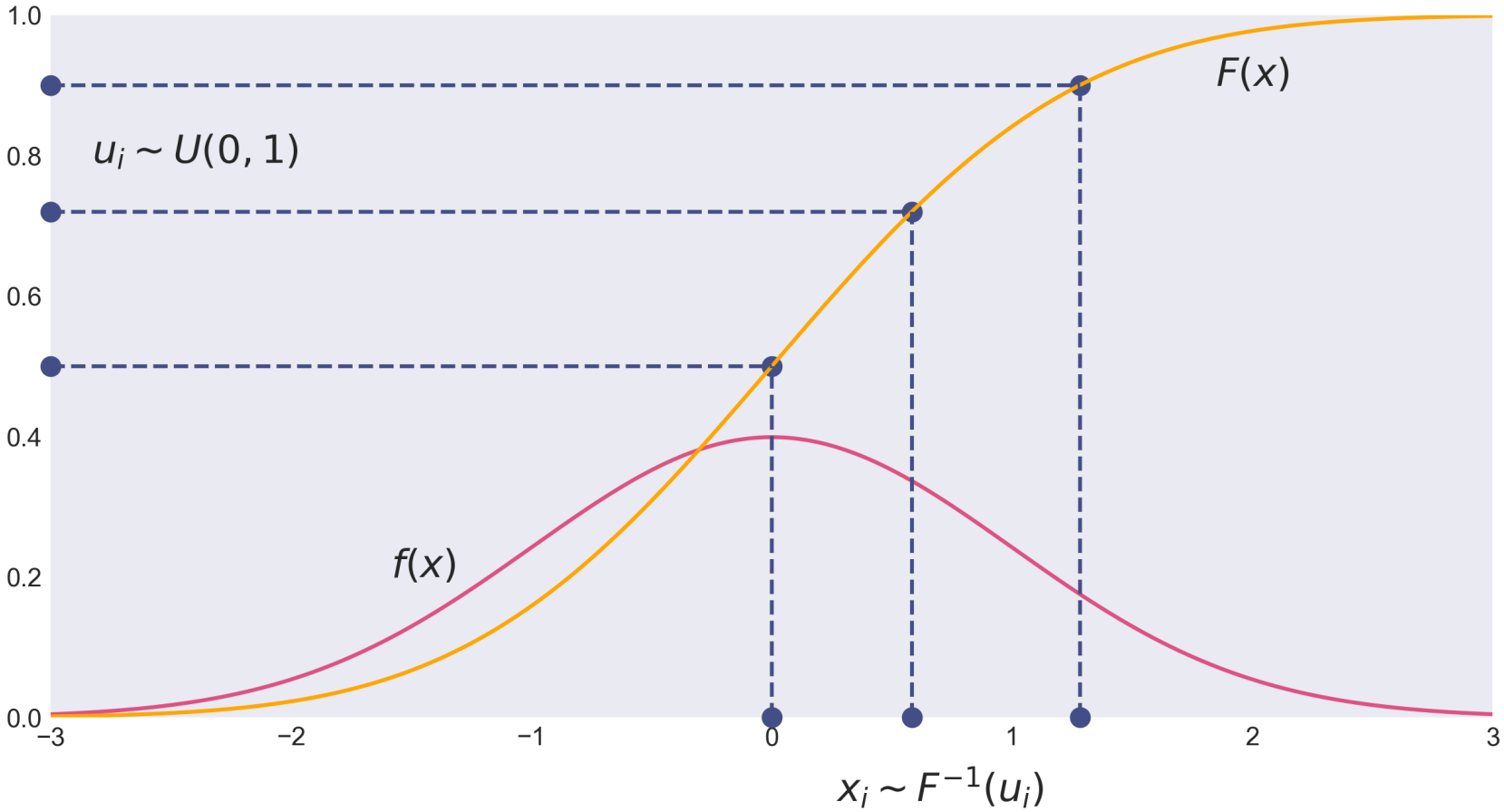


Table of Inverse CDFs for Common Distributions

Distribution	$F(x)$	$F^{-1}(p)$
$\mathcal{N}(\mu, \sigma)$	$\frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right]$	$\mu + \sigma\sqrt{2} \cdot \operatorname{erf}^{-1}(2p-1)$
$\mathcal{U}(a, b)$	$\frac{x-a}{b-a}$	$a + p \cdot (b-a)$
$\text{Exp}(\lambda)$	$1 - e^{-\lambda x}$	$\frac{-\ln(1-p)}{\lambda}$
$\text{Logistic}(\mu, s)$	$\frac{1}{1+e^{-\frac{x-\mu}{s}}}$	$\mu + s \ln\left(\frac{p}{1-p}\right)$

Rejection Sampling

Rejection Sampling

- What if we do not have a closed from CDF or ICDF?
- We can instead use Rejection Sampling!

Rejection Sampling: Algorithm

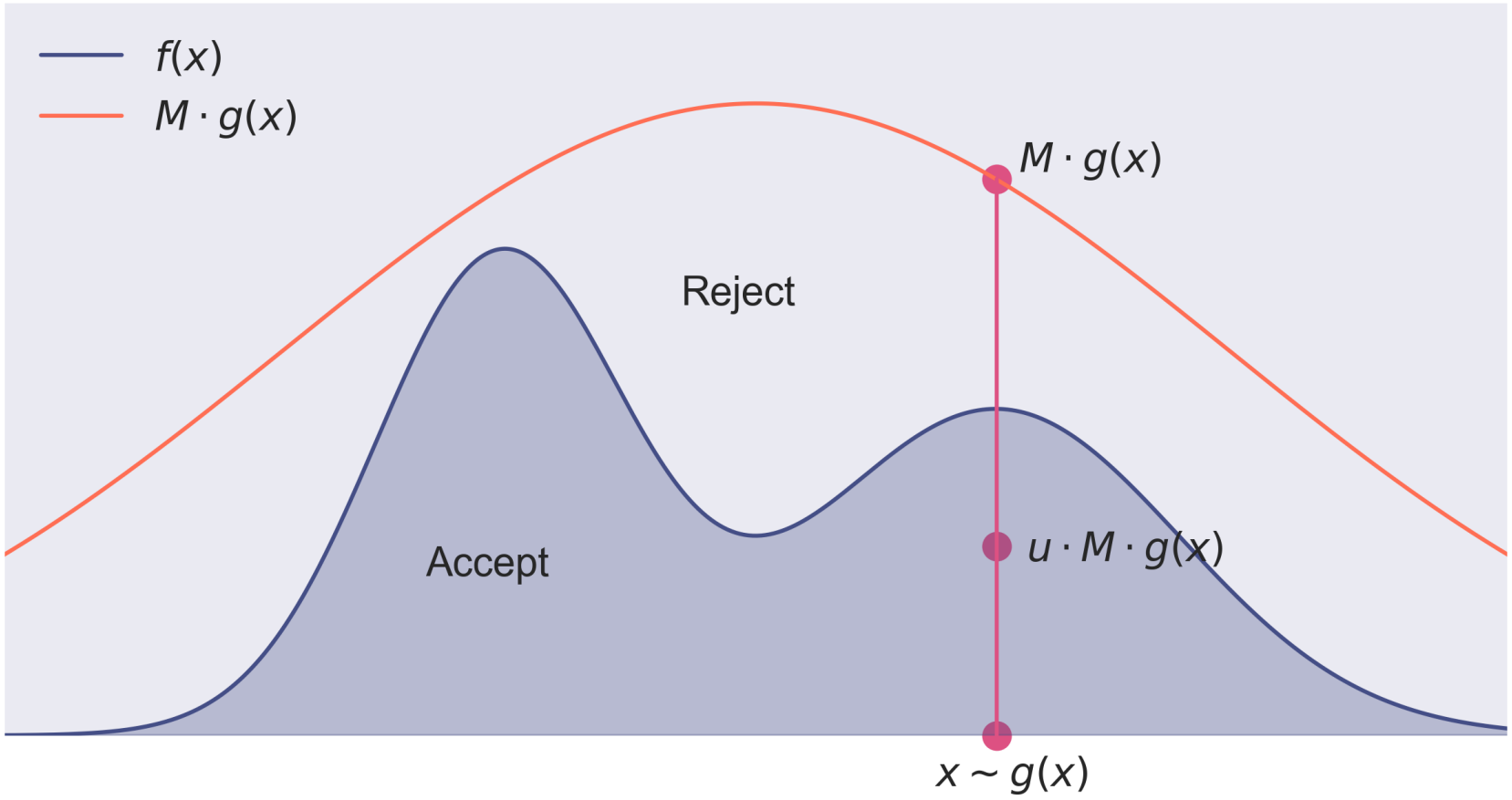
Algorithm 4 Rejection Sampling

Input: Model F, Proposal G, $M > 1$

1: Sample $x \sim G$
2: Sample $u \sim \text{Uniform}(0, 1)$
3: **if** $u < \frac{f(x)}{M \cdot g(x)}$ **then**
4: Accept x
5: **else**
6: Reject x
7: **end if**

Output: Accepted Samples

Rejection Sampling: Intuition



Rejection Sampling: Example

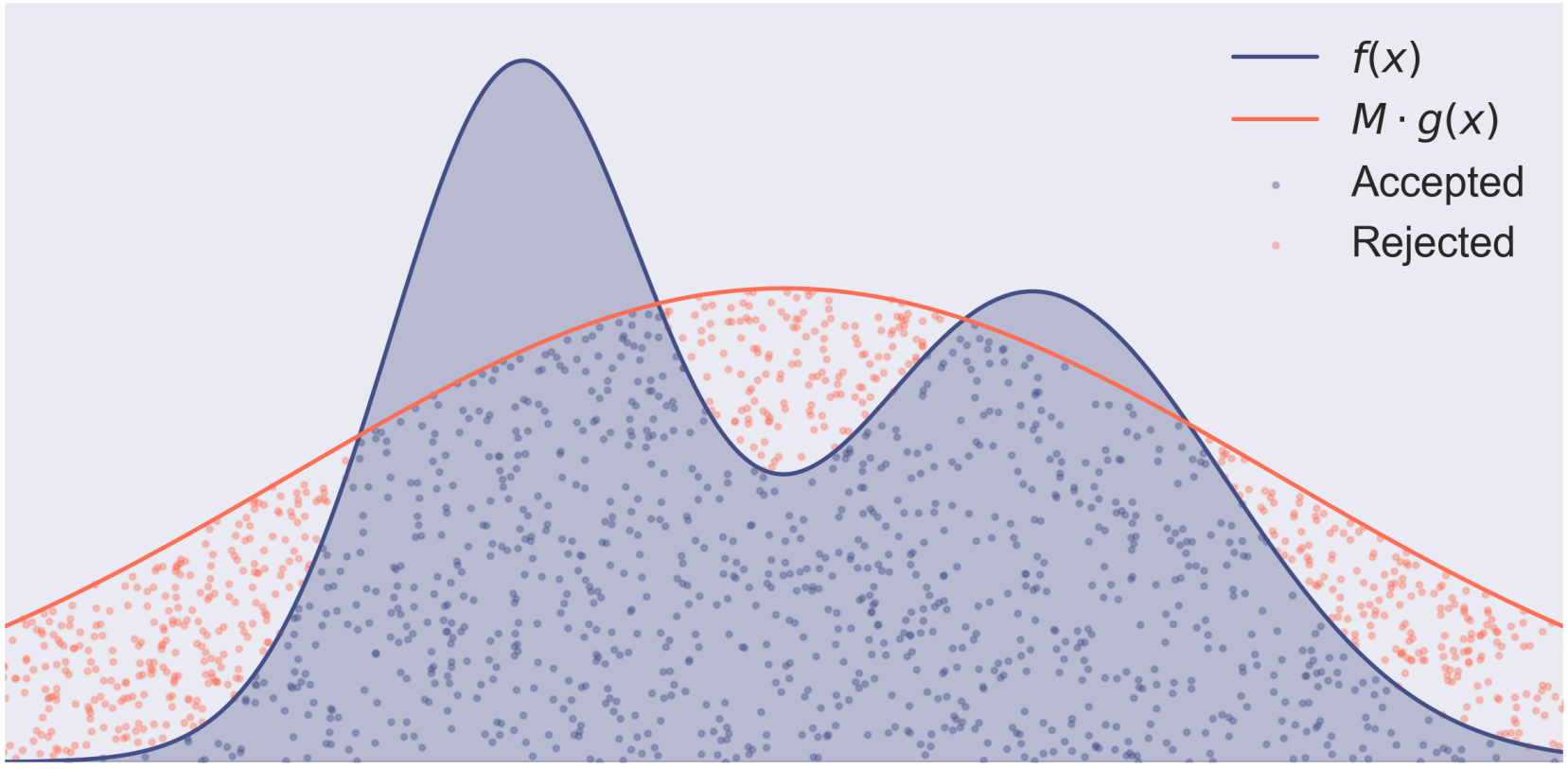


Figure: Rejection Sampling with $M = 1.3$

Rejection Sampling: Example

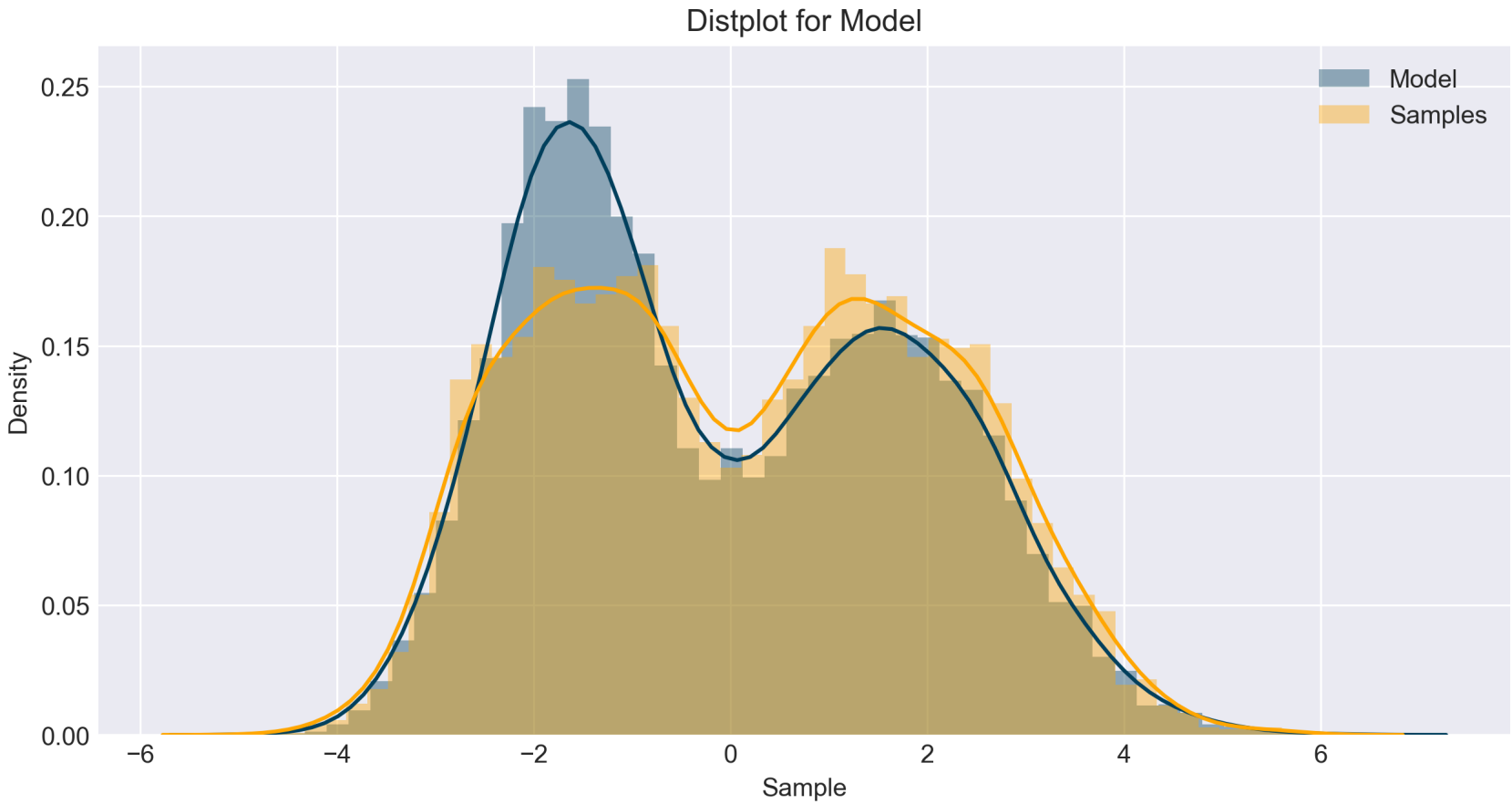


Figure: $M = 1.3$, $6,660$ Accepted Samples

Rejection Sampling: Example

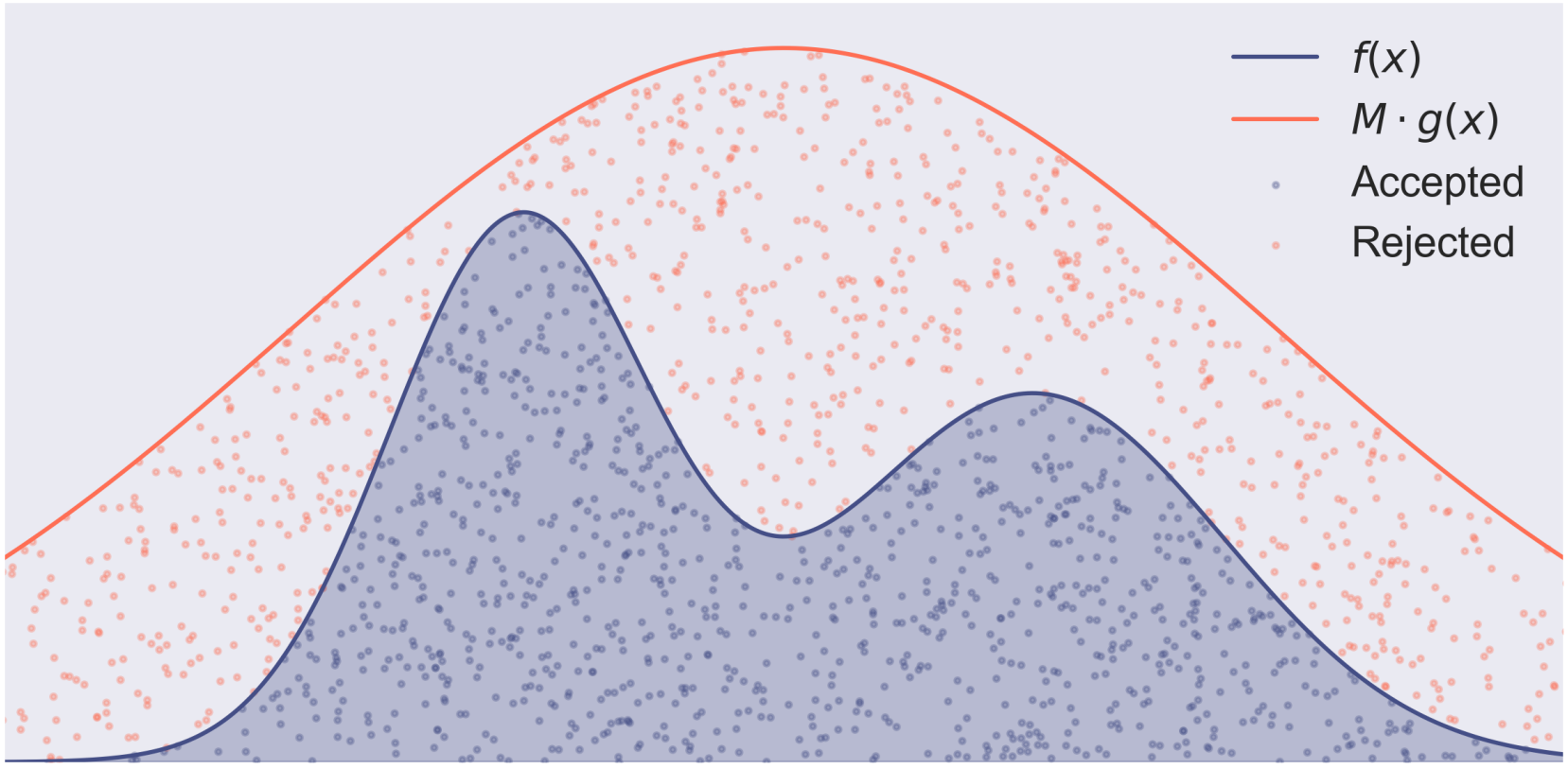


Figure: Rejection Sampling with $M = 2.5$

Rejection Sampling: Example

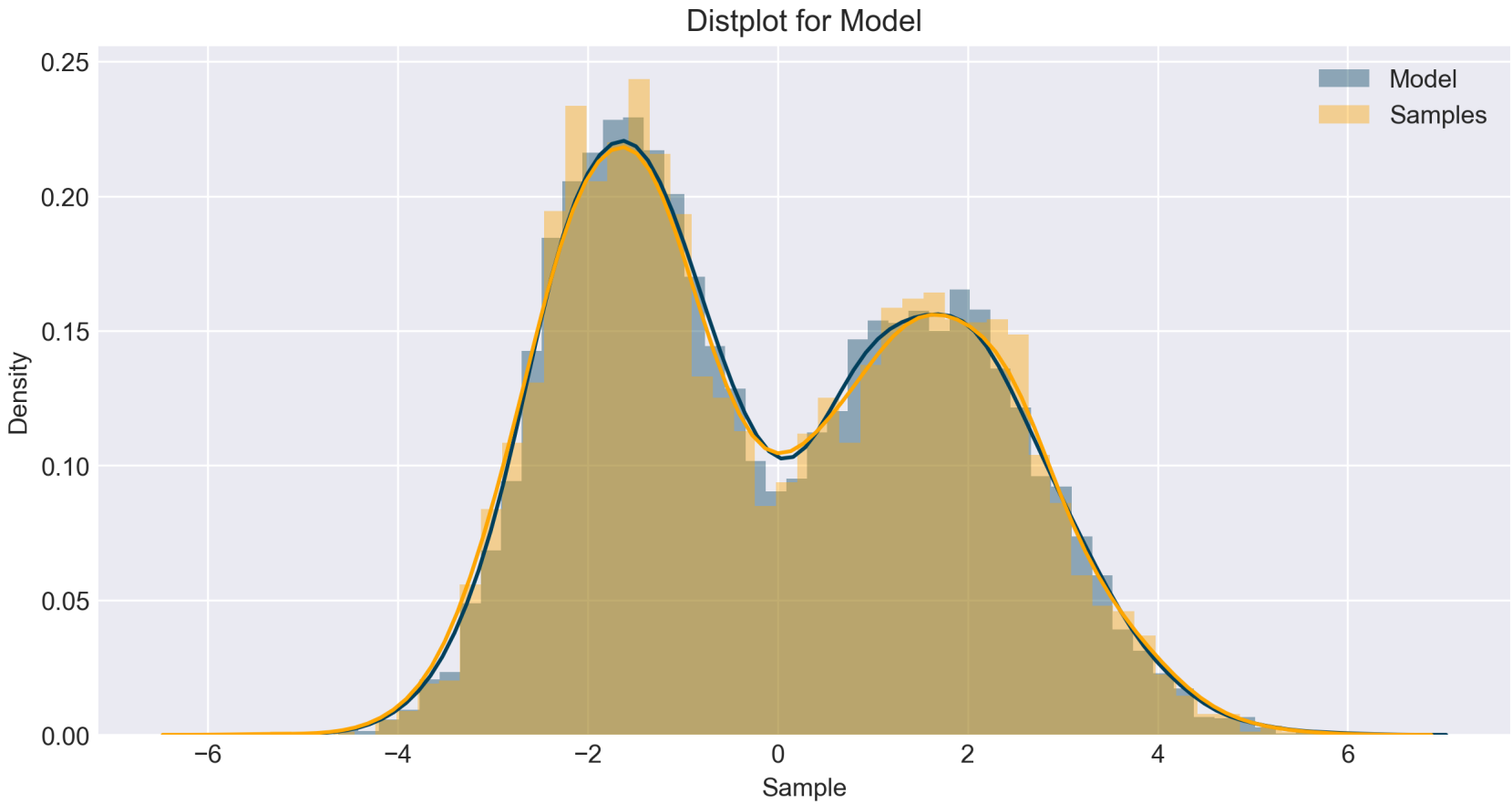


Figure: $M = 2.5$, $4,034$ Accepted Samples

Rejection Sampling: Example

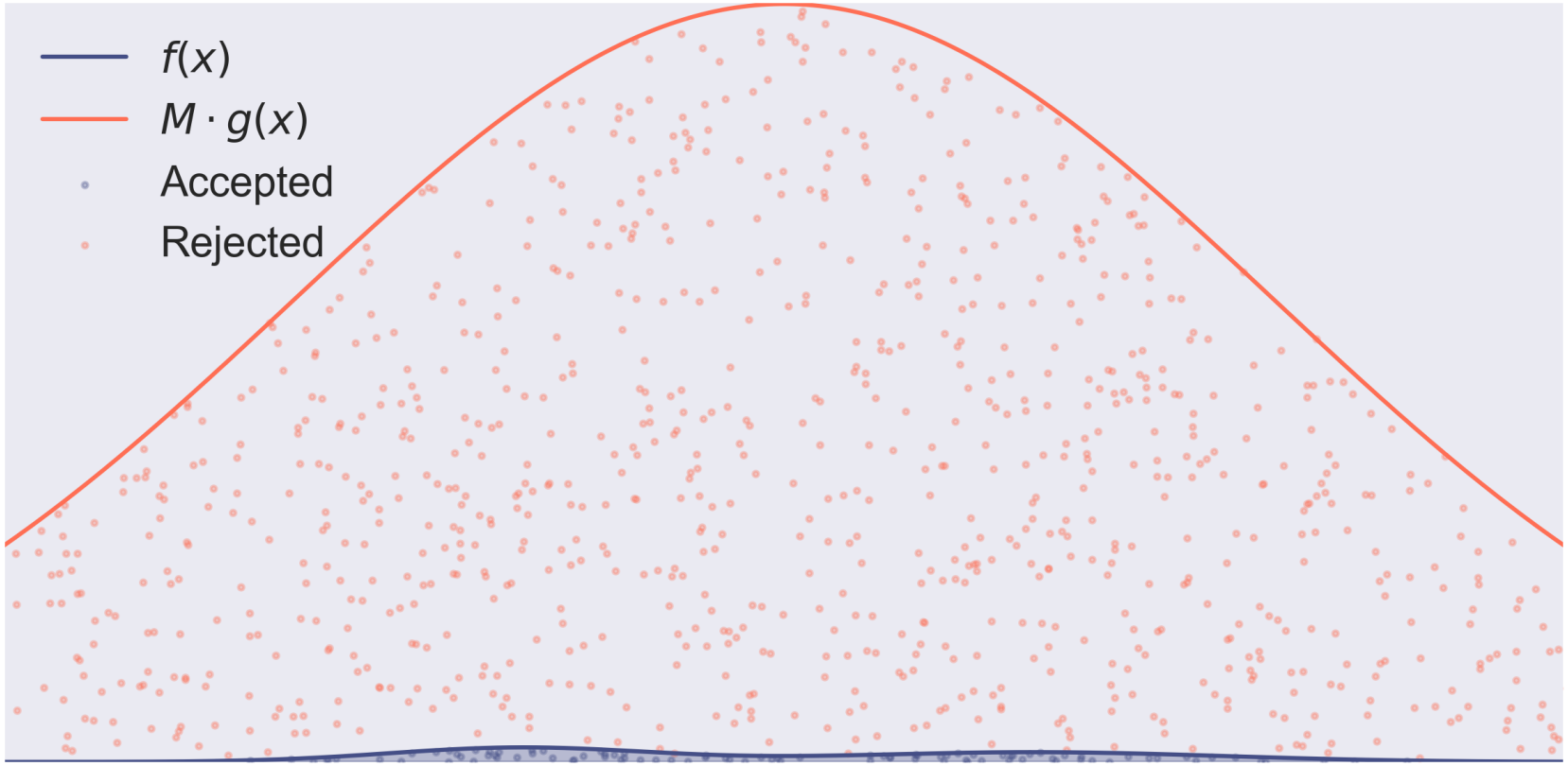


Figure: Rejection Sampling with $M = 100$

Rejection Sampling: Example

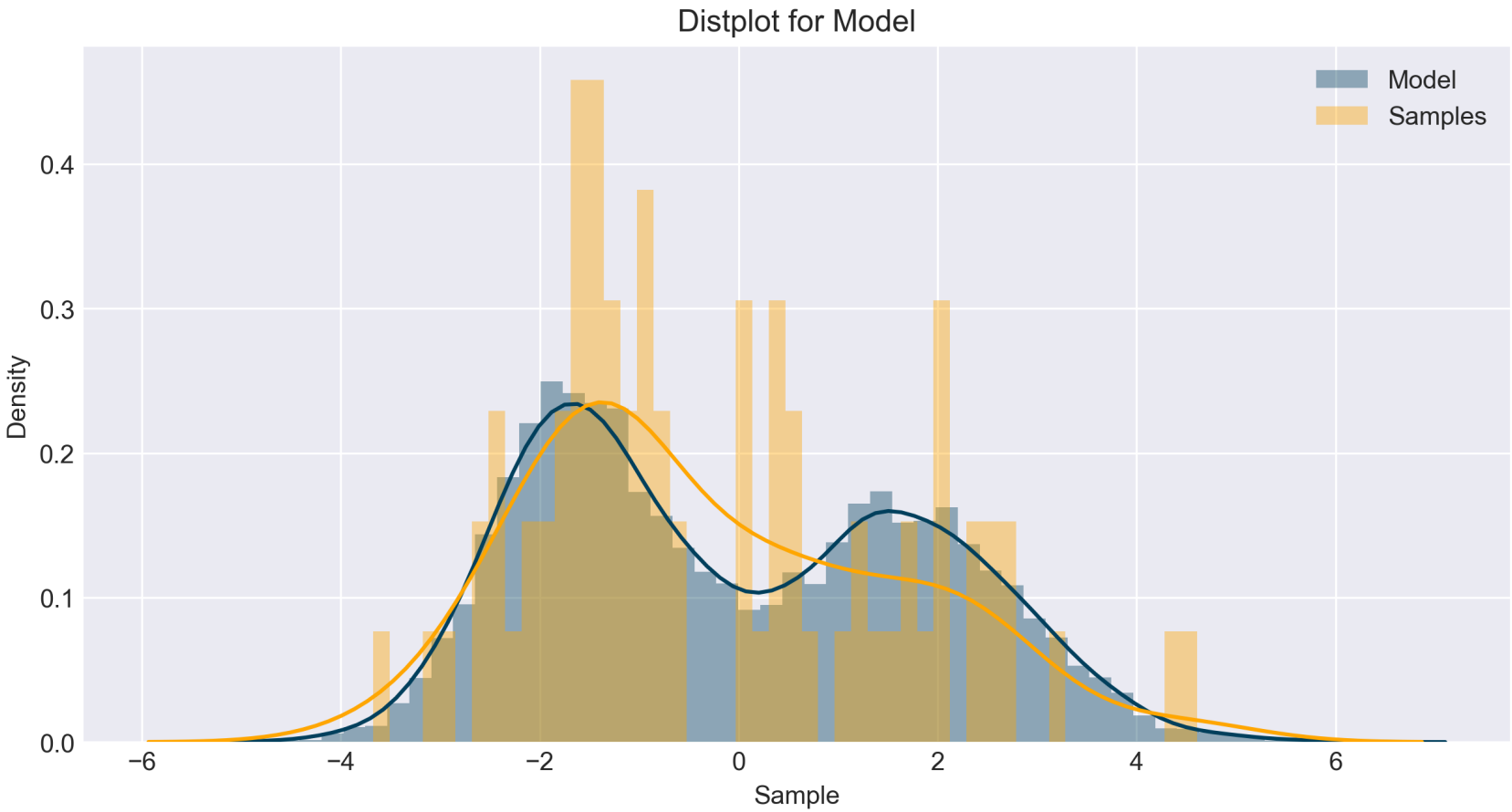


Figure: $M = 100$, 79 Accepted Samples

Rejection Sampling: Pros & Cons

- Pros:
  - Can be more efficient if the CDF is intractable
- Cons:
  - Tuning M can be difficult. Too high and we reject too many, too low and we under approximate our target
  - Very inefficient in higher dimensions

Markov Chain Monte Carlo (MCMC)

What is MCMC?

- Idea: Construct a Markov chain whose stationary distribution is the target density of interest, $f(x)$.
- The more steps we take in the chain, the better the approximation.
- This method works well with multi-dimensional continuous variables.

Metropolis-Hastings

Algorithm 5 Metropolis-Hastings Algorithm

Input: Model F, Proposal G

Initialize x_0
for $s = 0, 1, \ldots$ **do**
  Sample $x' \sim g(x'|x_s)$
  Compute acceptance probability

$$r = \min\left(1, \frac{f(x')}{f(x_s)} \frac{g(x_s|x')}{g(x'|x_s)}\right)$$

  Sample $u \sim \text{Uniform}(0, 1)$
  if $u < r$ **then**
    Accept x'
    Set $x_{s+1} = x'$
  else
    Reject x'
    Set $x_{s+1} = x_s$
  end if
end for

Output: Accepted Samples

- Construct a Markov Chain where we propose a new state x' from the current state x_s with probability $g(x'|x_s)$.
- After drawing a proposal x' we calculate an acceptance probability, and if accepted, update the state to x', else stay at state x_s.

Metropolis-Hastings: Example

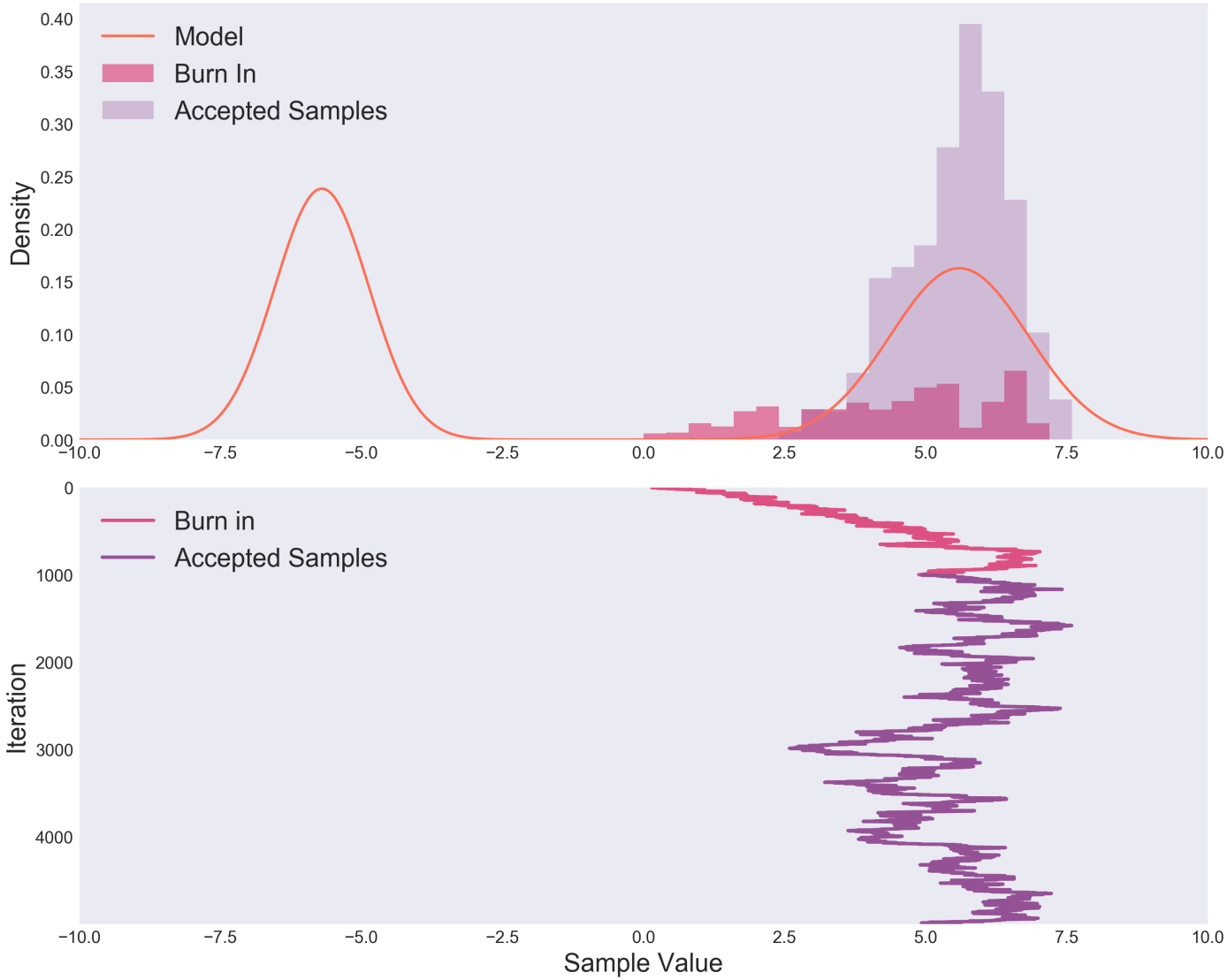


Figure: $\sigma^2 = 0.1$

Metropolis-Hastings: Example

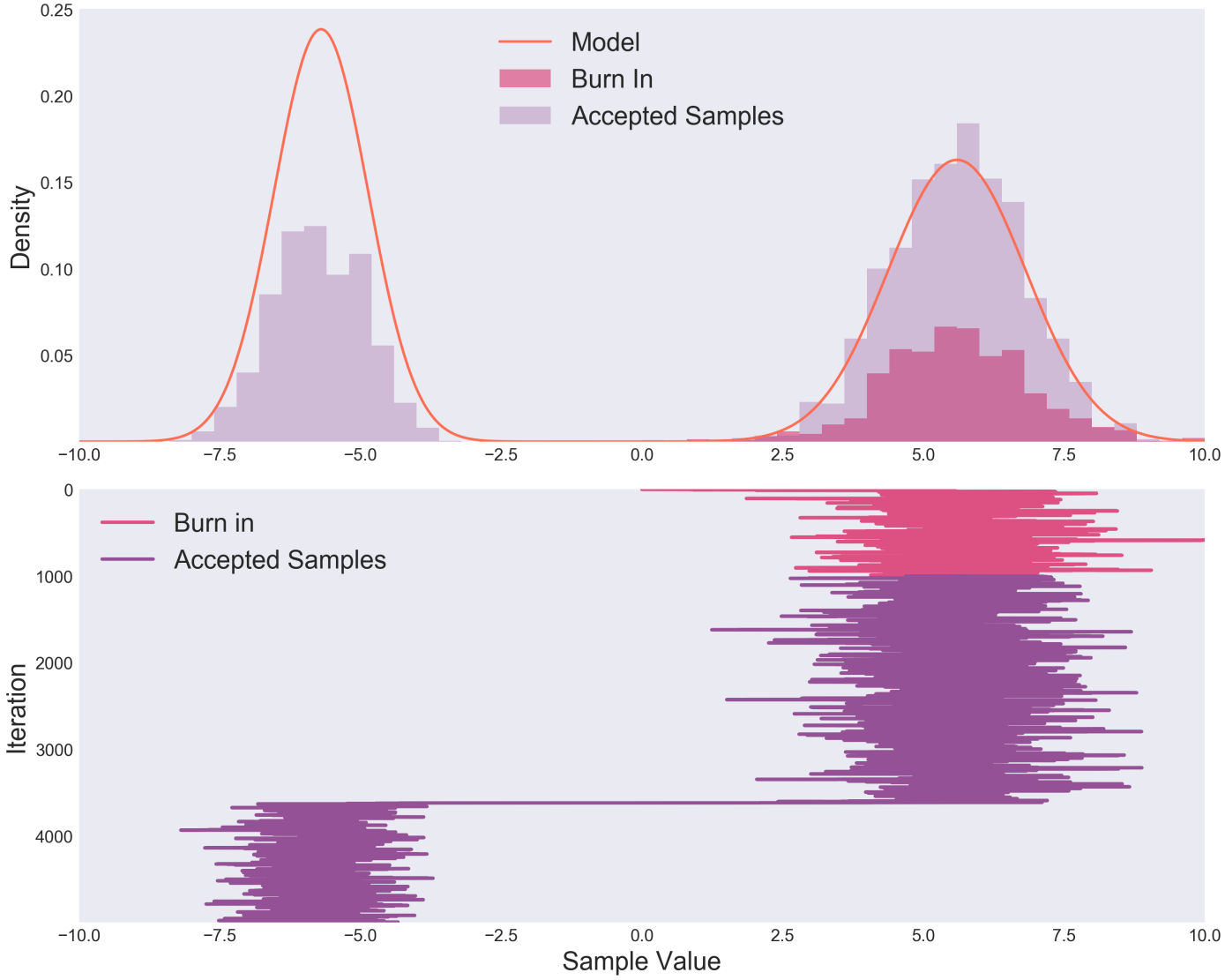


Figure: $\sigma^2 = 3$

Metropolis-Hastings: Example

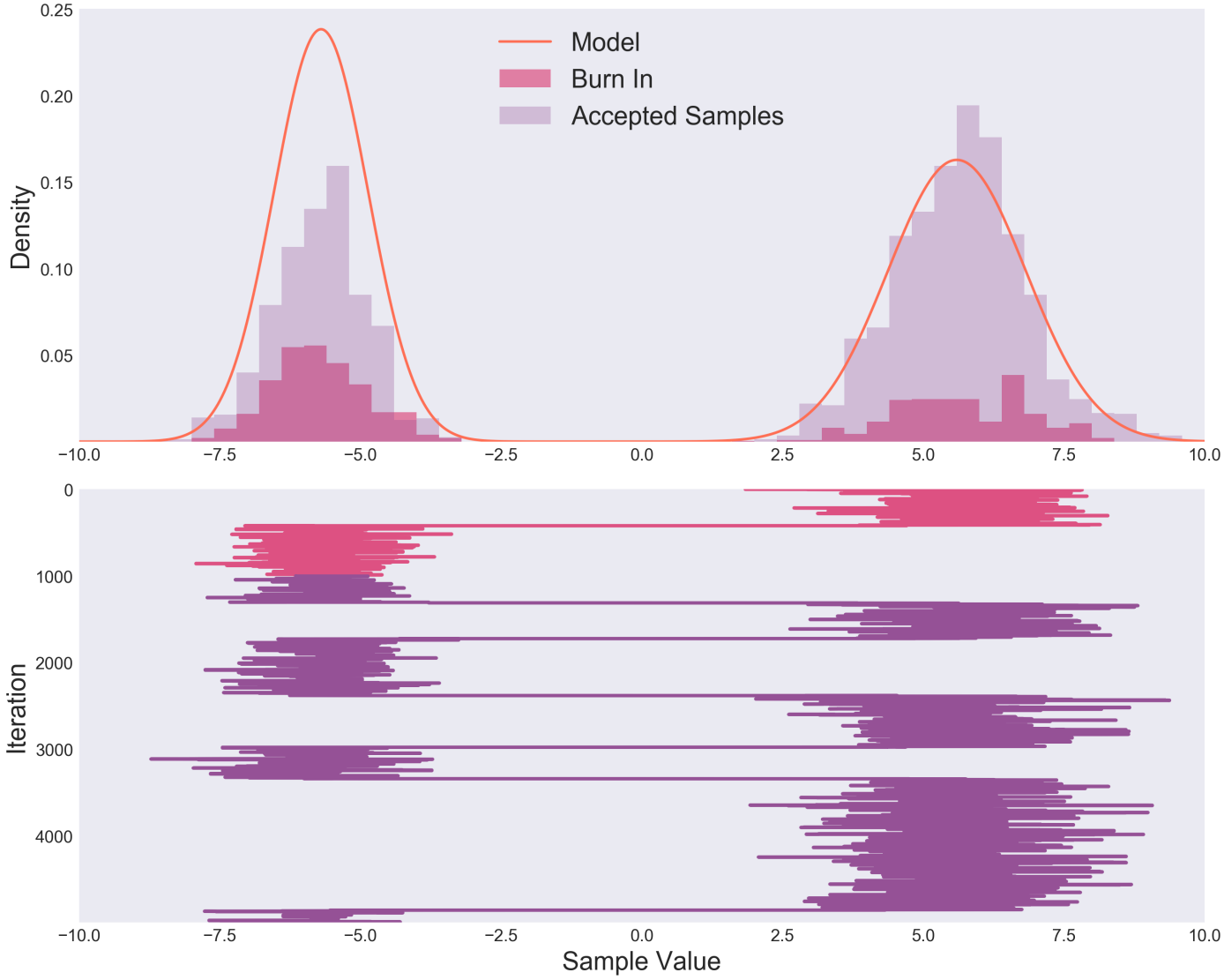


Figure: $\sigma^2 = 10$

Metropolis-Hastings: Example

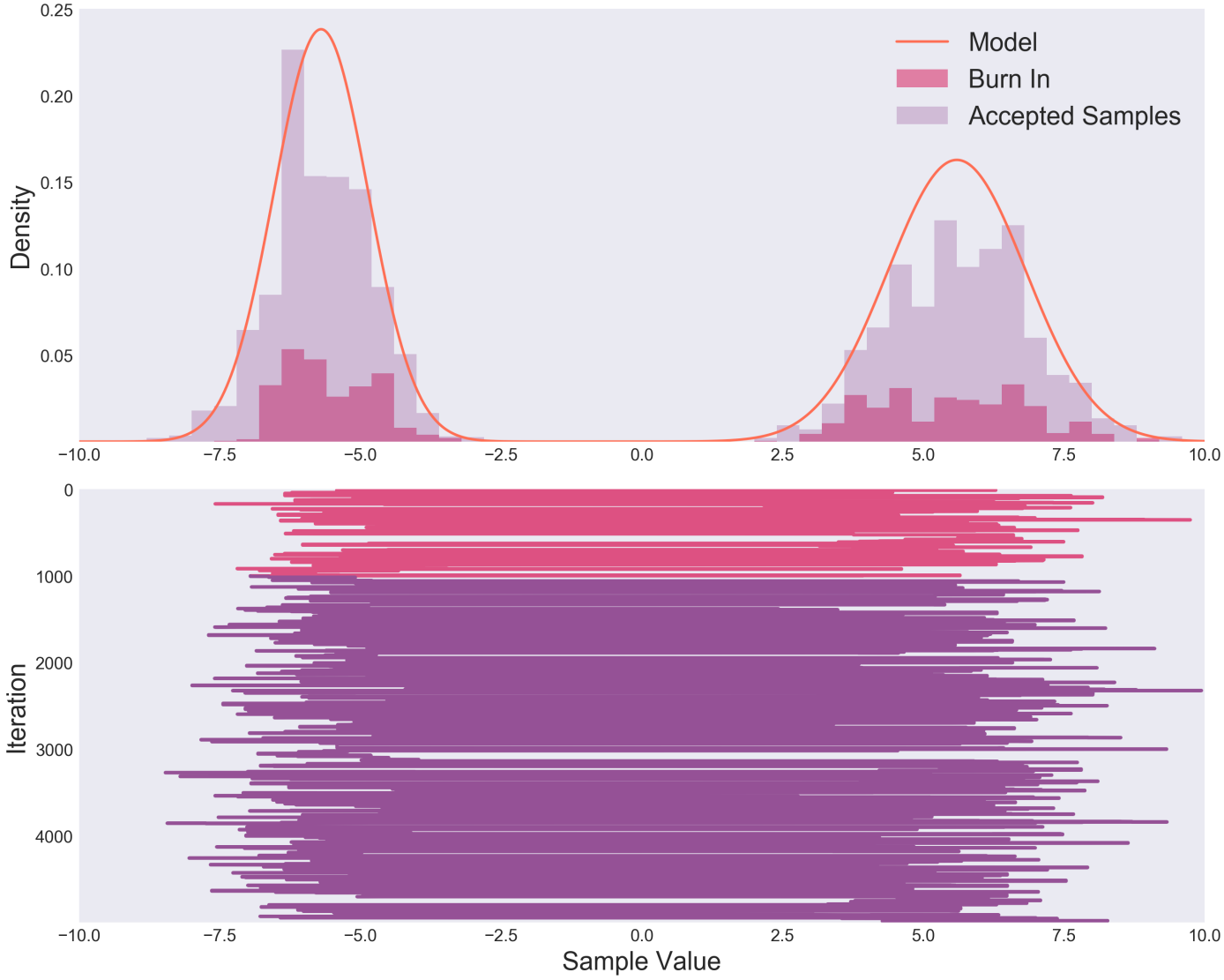


Figure: $\sigma^2 = 100$

Metropolis-Hastings: Key Terms to Know

- Acceptance Rates
  - Fraction of draws that are accepted
  - High acceptance rate $\rightarrow$ bad mixing
  - Low Acceptance rate $\rightarrow$ inefficient
  - Theoretical rates: 44% for one dimension, 23.4% as the dimension goes to infinity
- Chains
- Burn In
  - Allows the chain to "forget" its starting values and converge on areas of high probabilty
- Mixing
  - Allowing the chains to fully explore the state space, instead of collapsing in one peak

Variants of Metropolis-Hastings

(Random Walk) Metropolis Algorithm

- Uses a symmetric proposal distribution G, such that $g(x_s|x') = g(x'|x_s)$
- Our acceptance probability r is then

$$r = \min\left(1, \frac{f(x')}{f(x_s)}\right)$$

Metropolis Adjusted Langevin Algorithm (MALA)

- New states are proposed with Langevin dynamics

$$x' = x_s + \tau \nabla \log f(x_s) + \sqrt{2\tau}\xi_k$$

- Proposal probabilities are normally distributed as

$$g(x'|x_s) \sim \mathcal{N}(x_s + \tau \nabla \log f(x_s),\, 2\tau I_d)$$
$$g(x_s|x') \sim \mathcal{N}(x' + \tau \nabla \log f(x'),\, 2\tau I_d)$$

- Optimal acceptance rate is 57.4%

Metropolis Adjusted Langevin Algorithm: Example

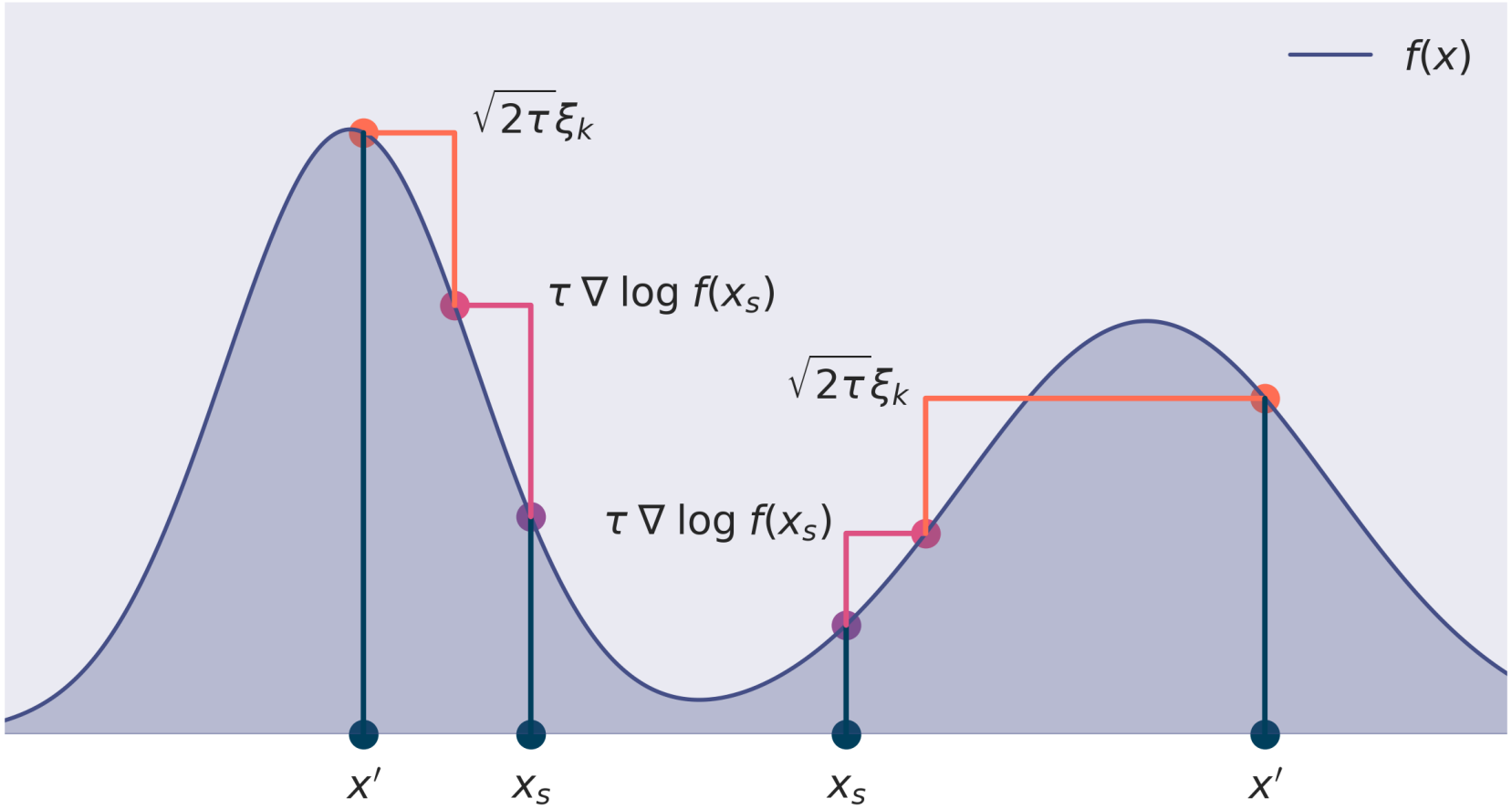


Figure: MALA: $\tau = 0.4$

Hamiltonian Monte Carlo

- Inspired by using a Hamiltonian dynamics evolution simulated using a time-reversible and volume preserving leapfrog integrator.
- Purpose was to reduce the correlation between successive sample states by proposing moves to distant states with high probability of acceptance.
- In simplier terms: flick a puck, wait, stop, then hit it again

Hamiltonian Dynamics

- For a system with state q, momentum p:

$$H(q,p) = U(q) + K(p)$$

$$U(q) = -\log f(q) \qquad K(p) = \sum_i \frac{p_i^2}{2}$$

- The time evolution of the system is defined by:

$$\frac{dp}{dt} = -\frac{\partial H}{\partial q} = -\frac{\partial U(q)}{\partial q} \qquad \frac{dq}{dt} = \frac{\partial H}{\partial p} = p$$

- We can update the system coordinates as follows (leapfrog):

$$q_{i+1} = q_i + \epsilon p_i \qquad p_{i+1} = p_i - \epsilon \frac{\partial U(q_{i+1})}{\partial q_{i+1}}$$

- Our acceptance probability for the current state (q_s, p_s) and candidate (q, p) is

$$r = \exp\left(H(q_s, p_s) - H(q, p)\right)$$

Hamiltonian Monte Carlo: Algorithm

Algorithm 6 HMC, Single Candidate Update

Input: Model F, Stepsize ϵ, Leapfrog Steps L, Current State x_s

Set $q = q_s$, $p \sim \mathcal{N}(0, 1)$, $p_s = p$
$p = p - \epsilon \frac{\partial U(q)}{\partial q} / 2$
for $l = 0, 1, \ldots L$ **do**
    $q = q + \epsilon \cdot p$
    $p = p - \epsilon \frac{\partial U(q)}{\partial q}$ except at end of trajectory
end for
$p = p - \epsilon \frac{\partial U(q)}{\partial q} / 2$
$p = -p$ to make proposal symmetric
Compute acceptance probability
$r = \exp\left(U(q_s) - U(q) + K(p_s) - K(p)\right)$
Sample $u \sim \text{Uniform}(0, 1)$
if $u < r$ **then**
    return q
else
    return q_s
end if

Hamiltonian Monte Carlo: The Leapfrog Path

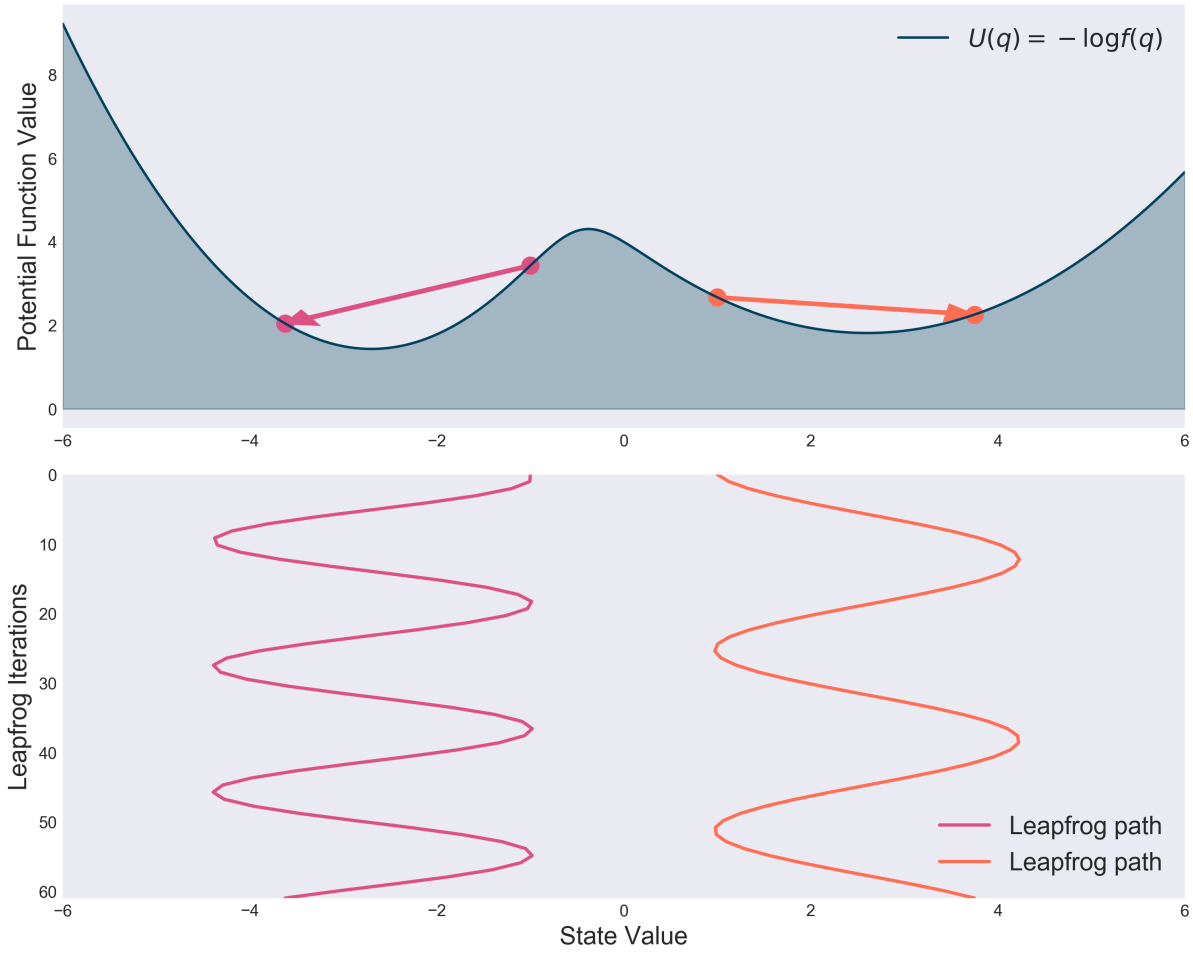


Figure: Leapfrog Paths: $\epsilon = 0.3$, $L = 60$

Hamiltonian Monte Carlo: Tampering to Overcome Energy Barriers

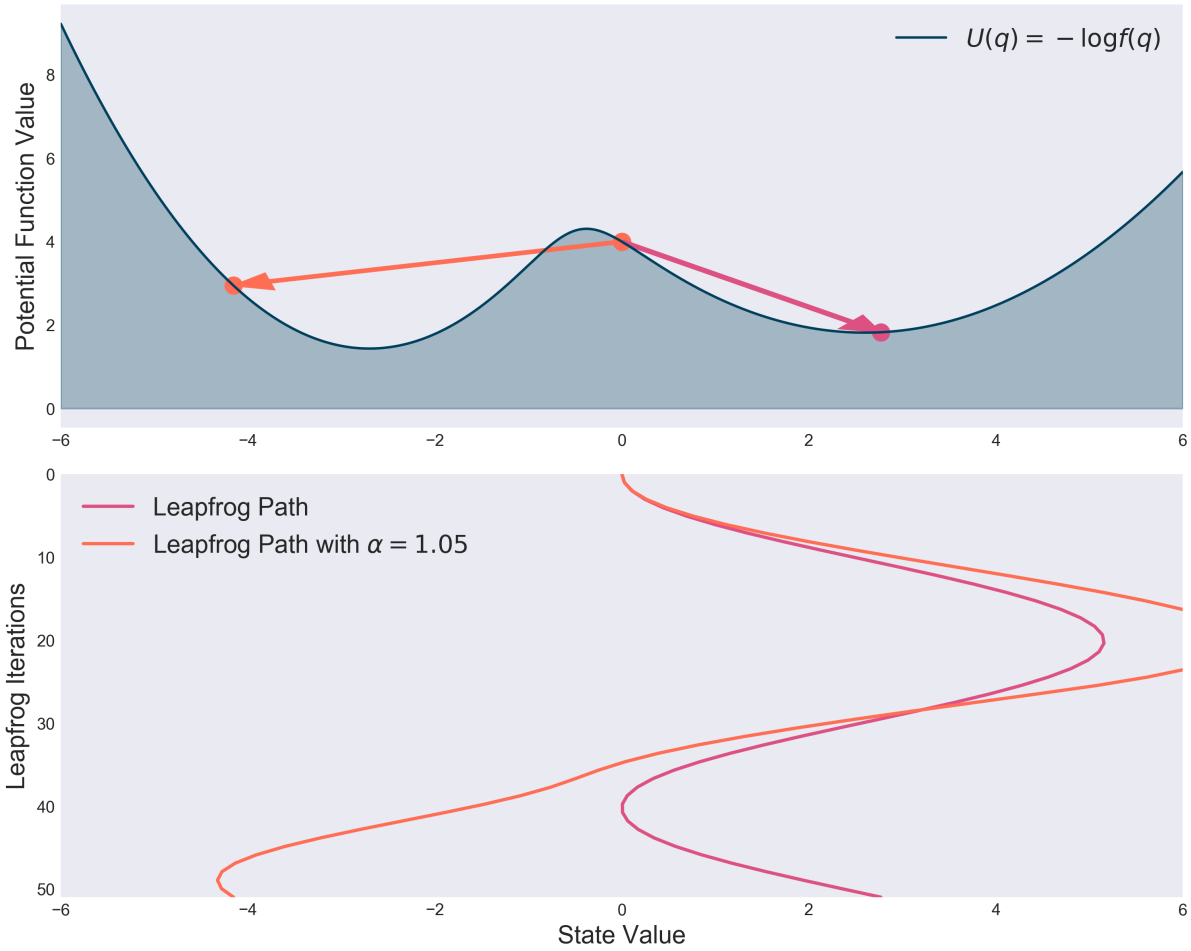


Figure: Leapfrog Paths: $\epsilon = 0.2$, $L = 50$, $\alpha = 1.05$

Hamiltonian Monte Carlo: Example

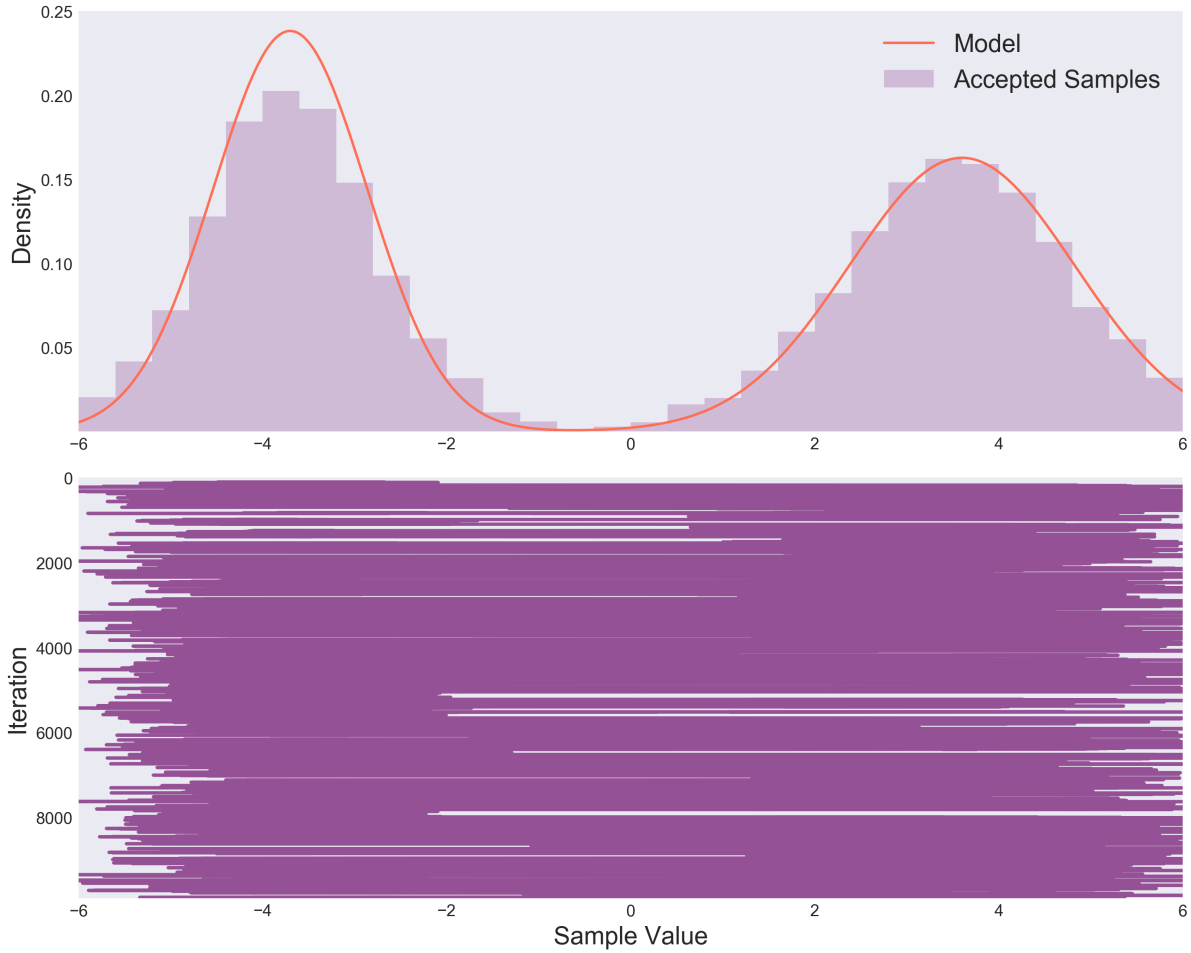


Figure: HMC: $\epsilon = 0.6$, $L = 40$, $\alpha = 1.05$, 100 burn in iterations, 10,000 epochs

Hamiltonian Monte Carlo: Considerations

- Distances between points are large, thus requiring less iterations
- Mostly accepts new states, more efficient even with the leapfrog "price"
- Tuning leapfrog steps can be difficult:
  - Small $L \rightarrow$ random walk behavior
  - Large $L \rightarrow$ wasted computation
- Has trouble sampling from distributions with isolated local minimums (lack of energy to cross the energy barrier)
- Optimal acceptance rate is 65%
- HMC Interactive Demo

No-U-Turn Sampler (NUTS)

- Removes the need to step the leapfrog step L in HMC
- Uses a recursive algorithm to build a set of likely candidate points
- As efficient as a well tuned HMC method

Gibbs Sampling

-

Analysis of Error

- How many samples S does it take to approximate the target distribution "well"?
- Answer: Use the Hoeffding bound

$$Pr\left(\hat{p}(x) \notin [p(x) - \epsilon, p(x) + \epsilon]\right) \leq 2e^{-2S\epsilon^2}$$

- For the number of samples S, an error bound ϵ with probability $1 - \delta$, we can solve:

$$2e^{-2S\epsilon^2} \leq \delta$$

$$S \geq \frac{\log(2/\delta)}{2\epsilon^2}$$

Analysis of Error

- We can also use the Chernoff Bound relative to the true value $p(x)$
- However, this is dependent on $p(x)$, which is not always known.

$$Pr\left(\hat{p}(x) \notin [p(x)(1-\epsilon), p(x)(1+\epsilon)]\right) \leq 2e^{-Sp(x)\epsilon^2/3}$$

$$S \geq 3\frac{\log\left(2/\delta\right)}{p(x)\epsilon^2}$$

Libraries and Tools

- PyMC3
- TensorFlow Probability
- Stan
- RStan
- R mcmc

Refrences & Further Reading

- Machine Learning: A Probabilistic Perspective by Kevin Murphy
- Monte Carlo Methods in Financial Engineering by Paul Glasserman
- Probabilistic Graphical Models: Principles and Techniques by Daphne Koller and Nir Friedman
- Sampling Lecture from my PGM Professor Daniel Malinsky
- MCMC Using Hamiltonian Dynamics by Radford M. Neal
- Hamiltonian Monte Carlo Explained by Alex Rogozhnikov
- The Markov-chain Monte Carlo Interactive Gallery by Chi Feng
- NUTS Paper by Hoffman and Gelman